WELL-ESTABLISHED DOCUMENT INTERCHANGE FORMATS

JEAN-MARIE DE LA BEAUJARDIÈRE

Xerox Palo Alto Research Center

ABSTRACT

A number of standards exist to facilitate the interchange of electronic documents between different computer systems. Though they each have their place and have proved very useful, none has reached the level of generality and completeness required to ensure an exchange of information without loss, especially in round-trip interchange. This paper takes a look at four well-known interchange formats (DIF, DCA, SGML and ODA) and compares the ways they handle the same sample document.

1. Introduction

The technology of document interchange between incompatible computer systems is still in its infancy, especially in the case of round-trip interchange. Both sides of the interchange, the sending side and the receiving side, are too weak to perform a completely useful exchange:

- On the sending side, there is not yet a fully defined interchange format which captures and describes the complexity of composite documents. The contents and structure of electronic documents may vary from simple text with limited typographical variations, to data-driven graphics, color images, voice recordings and other elaborate objects. No language exists today to describe all these objects in a consistent way. The most recent proposal, ODA [1], is much talked about, but so far there has been more talk than solid implementations.
- On the receiving side, it is not clear how a system should handle a document object which it does not support. Were a fully complete interchange language to exist, what would a lowly PC do when presented with a voice recording or a complex chemical equation? Today's solution is to discard the embarrassing object or some of its attributes. This cannot be the long-term solution.

Figure 1: a composite (text & graphics) document to interchange

1. **The Beginning.**

   John "Le Naïf" Léger paced the room, reading and rereading the incredible offer that came in the mail this morning.

   It was too good to be true...

2. **The Middle.**

   ... They went to see that "land" they bought in Florida ...

3. **The End.**

   ... Jane said: "I will come back as soon as chicken start growing teeth." She left the room, leaving John holding the mortgage.

Defining a general interchange format and specifying the proper behavior of systems that do not implement its full generality are difficult problems. Quite a number of systems have successfully provided solutions to a subset of the problems, limiting themselves to the interchange, for instance, of structured graphics or simple text and its logical structure. This paper takes a look at several interchange formats and compares the way they handle a small example, shown in Figure 1. An interesting comparison of two of the products discussed here, SGML and ODA, can be found in [Joloboff86].

2. DIF

DIF stands for Document Interchange Format. It is an encoding developed at the behest of the Department of the Navy by the National Bureau of Standards and a group of software manufacturers [Knoerdel84]. It is used to exchange text, not graphics or complex structures, and formatting instructions across a wide variety of word processors and publishing systems.

A DIF document is a linear stream of 7-bit ASCII bytes. Control codes, prefixed by an Escape character, are interspersed with the text. Thirteen DIF functions are actually codes already defined in other standards ([ANSI79], [ANSI83] and [ISO83]). Twenty-nine additional functions are encoded according to the escape sequence mechanism provided by [ANSI79].

Figure 2 represents a DIF encoding of Figure 1. The conversion program had no choice but to discard the graphics (the three faces). It replaced the proportionally-spaced typeface by an undefined fixed-pitch font of the closest size (the only choices being 12- and 10-pitch). Finally, it dropped the accents

Figure 2: legible representation of DIF encoding of Figure 1

ESC [1$K	start document
ESC [0 SP K	10-pitch
ESC [1$J	start page
ESC [1m	emphasis on
1.HTThe Beginning.	
ESC [22m	emphasis off
ESC E	end paragraph
ESC [14$F	temporary left margin
John "Le Naif" Leger paced the room, reading and rereading the	
ESC [4m	underscore on
incredible	
ESC [24m	underscore off
offer that came in the mail this morning.	
ESC E	end paragraph
ESC [14$F	temporary left margin
It was too good to be true...	
ESC E	end paragraph
ESC [1m	emphasis on
2. HT The Middle.	
ESC [22m	emphasis off
ESC E	end paragraph
ESC [14$F	temporary left margin
... They went to see that "land" they bought in Florida ...	
ESC E	end paragraph
ESC [1m	emphasis on
3. HT The End.	
ESC [22m	emphasis off
ESC E	end paragraph
ESC [14$F	temporary left margin
... Jane said: "I will come back as soon as chicken start growing teeth."	
She left the room, leaving John holding the mortgage.	
ESC [0$J	end page
ESC [0$K	end document

over accented letters and replaced the opening and closing quotes by neutral quotes, because of the limitations of the 7-bit ASCII code set.

3. DCA

DCA stands for Document Content Architecture. It is an encoding promoted by IBM as a common document format for its word processors. Since a large number of software companies jumped onto the band-wagon and provided document conversion from and to DCA, that format became a *de facto* inter-

change standard. DCA comes in two flavors: Final-Form-Text DCA [IBM85], not amenable to editing and not discussed further here, and Revisable-Form-Text DCA [IBM83], encoding documents which may be edited.

The DCA document model is typewriter-oriented. Ever present is the image of a printing assembly moving horizontally and vertically over a sheet of paper. Superscripting, for instance, is seen as an instruction to the printing assembly to "move up a fraction of a line," rather than being a property attached to a character. Consistent with the typewriter model, a DCA document contains character text only; it does not contain images, structured graphics, bar charts, tables (except primitive ones), equations, etc.

A DCA document is internally represented as a stream of EBCDIC bytes. The stream is divided into *Structures*, some containing the document text intermingled with control information, and some containing pure control information governing the appearance of the document. There are structures subdivided into *Self-Identifying Parameters* which act like buckets to collect a set of related control information. The text itself may be interspersed with *One-byte Controls* and *Multibyte Controls,* some of which may contain Self-Identifying Parameters. In light of the paucity of features of DCA, it is not clear what is gained by encoding the control information in five different ways: fixed structures, parameterized structures, fixed multibyte controls, parameterized multibyte controls, and one-byte controls.

As in DIF's case, a program converting the sample document into DCA has no choice but to discard the graphics. It has to map the typeface into one of the typefaces registered by IBM (with the attending problem of font substitution). The accented letters are correctly mapped since they have a code in the EBCDIC character set 337. The opening and closing quotes are replaced by left angle and right angle quotes, rather than neutral quotation marks. Figure 3 represents the Revisable Form stream corresponding to the sample document. For brevity's sake, a number of DCA codes have been dropped and, for legibility, the remaining ones have been replaced by explanatory text.

4. SGML

SGML stands for Standard Generalized Markup Language. An outgrowth of IBM's Generalized Markup Language, SGML is an international standard developed by ANSI and ISO [ISO86a]. SGML has a very different approach to document interchange. It divides the world of makers of documents into authors, who know the meaning of their writings, and publishers, who know how to layout and present documents. (Obviously, SGML was invented before the advent of desktop publishing).

Figure 3: DCA Revisable-Form encoding of Figure 1
(DCA codes replaced by legible text and comments)

Structure FUP (5 bytes)	format unit prefix
Structure DP (26 bytes)	document parameters
Structure FUP (5 bytes)	format unit prefix
Structure PMF (61 bytes)	primary master format
Structure PIP (19 bytes)	page image parameters
Structure LP (30 bytes)	line parameters
Structure TP (7 bytes)	tab parameters
Structure TUP (11 bytes)	text unit prefix
Structure BT (694 bytes)	body text
Multibyte SFG (7 bytes)	set font attributes
Multibyte BES (3 bytes)	begin emphasis
1.HTThe Beginning.RCR	
Multibyte EES (2 bytes)	end emphasis
ITJohn «Le Naïf» Léger paced the room, readingCRE	
and rereading the	
Multibyte BUS (4 bytes)	begin underline
incredible	
Multibyte EUS (2 bytes)	end underline
offer that came in CRE	
the mail this morning.RCR	
ITIt was too good to be true...RCR	
Multibyte BES (3)	begin emphasis
2.HTThe Middle.RCR	
Multibyte EES (2 bytes)	end emphasis
IT ... They went to see that «land» they bought inCRE	
Florida ...RCR	
Multibyte BES (3)	begin emphasis
3.HT The End.RCR	
Multibyte EES (2 bytes)	end emphasis
IT... Jane said: «I will come back as soon asCRE	
chicken start growing teeth.» She left the room, CRE	
leaving John holding the mortgage.	
PE	page end
Structure EUP (5 bytes)	end unit prefix
Structure BT (6 bytes)	body text
PE	page end

Authors know which portion of text is dependent on another, which fragments should be emphasized, what is a quote and what is a qualifier. Authors know the different meanings they attach to the quotes in the strings: "Le Naïf" (nickname), "land" (irony) and "I will come back..." (verbatim). SGML allows authors to note such differences of intent. On the other hand,

since authors are not publishers, SGML does not allow them to specify the rendering of these differences.

Publishers know how to make the ink on paper look good: what typeface is appropriate for the document's contents, how much space is adequate around each paragraph, where to place pictures and footnotes, etc.

Publishers receive SGML files from authors, of course. The files contain the authors' text, the markups made by authors to annotate the logical structure of the text, and some "meta-language instructions" which describe the markups themselves (defining, for instance, the actual characters used in the file to enclose the markup strings). The meta-language instructions enable the publishers' programs to verify the coherence of the authors' markups.

Publishers also receive from authors verbal or written instructions (that is, non-standard and non-computerized instructions) on the meaning of the markups. Aye, there's the rub when it comes to document interchange with SGML. Authors can attach meaning in many different ways to their text: publishers cannot know and SGML cannot standardize these ways in advance.

Consider, for instance, the three types of quotes in the sample document. They could be encoded in SGML as

John <n>Le Naïf</n> Léger
that land they bought
<v>I will come back as soon as chicken start growing teeth</v>

The author must explain to the publisher the intent behind the markups <n>, and <v>, so that the publisher can decide how to render them in print. Having so decided, the publisher calls in an SGML application programmer, who writes a program to process the <n>s, s and <v>s according to the publisher's instructions. A new SGML application must be developed for every new type of document. Figure 4 is an acceptable SGML input to produce Figure 1.

The lines beginning with ENTITY or ELEMENT up to the closing angle bracket are the SGML declarations. Assume that we are typing the document on a terminal without accented letters; the ENTITY declarations give a symbolic name to the accented letters used in the text and contain a comment to help the publisher understand the meaning of the declaration. The line numbered 1 declares that the document is made up of objects labelled h, each containing one or more objects labelled p and zero or one object labelled pict. A pict, if present, appears after or before the p's; the string "- -" means that the markups <h> and </h> are required. Line 2 declares that a p consists of characters or e's or n's or i's or v's, in any number and combination. Line 3 says that the contents of a pict is not to be found in the SGML stream; all that

Figure 4: SGML markup for Figure 1

```
<!ENTITY     eacute     SDATA     "-- e, acute--">
<!ENTITY     idier      SDATA     "-- i, dieresis--">
<ELEMENT
1  h     - -  (p+|pict)                  --header--
2  p     - O  (#CDATA|e|n|i|v)+          --paragraph--
3  pict  - O  NONE                       --picture--
         name                CHARS      REQUIRED
4  e     - O  (#CDATA)                   --emphasis--
5  n     - O  (#CDATA)                   --nickname--
6  i     - O  (#CDATA)                   --irony--
7  v     - O  (#CDATA)                   --verbatim--
>
<h>The Beginning.
<pict name="SmilingFace">
<p>John <n>Le Na&idier;f</n> L&eacute;ger paced the room,
reading and rereading the <e>incredible</e> offer that came in
the mail this morning.<p>It was too good to be true...</h>
<h>The Middle.
<pict name="SadFace">
<p>... They went to see that <i>land</i> they bought in Florida
...</h>
<h>The End.
<pict name="AngryFace">
<p>... Jane said: <v>I will come back as soon as chicken start
growing teeth</v>. She left the room, leaving John holding the
mortgage.</h>
```

appears with a pict object is a required character string spelled name; we will come back to this in a minute. The strings "- O" in lines 2 through 7 mean that the beginning markups must be present, but the end markups may be omitted when implied by the context (e.g. isn't necessary before <h> or).

As we have just seen, the contents of the pictures come from somewhere else. Each picture might be shipped to the publisher in a file with a name matching the name attribute of the pict element, or the pictures might exist only on paper and need to be pasted in manually. The author, the publisher and the SGML application programmer will agree among themselves on how to manage this process and make everything fall in place. The vagueness of the procedure is one of the reasons why SGML is looking toward ODA ([ISO86a], discussed below) as a kind of packaging tool to transport all the data pertaining to a composite document together.

5. ODA

ODA (Office Document Architecture) and ODIF (Office Document Interchange Format) are a standard developed by ISO to facilitate the interchange of documents [ISO86a]. ODA describes the concepts embodied in the standard in English prose and illustrations, while ODIF describes the bits and bytes of the actual encoding format.

Three aspects of ODA documents are discussed here: *layout structure, logical structure* and *content.*

5.1. *Layout Structure*

The layout structure tells application programs where to place the document elements on the presentation surface. The layout structure is a hierarchy of objects with a *document layout root* at the top, then *page sets*, *pages*, multiple levels of rectangular *frames*, and one level of rectangular *blocks* at the bottom. Page sets and frames are optional; the minimum layout structure consists of a root, a page and a block. Frames may overlap on the page; blocks within the same frame may not. Blocks point to content portions of a single type (such as text or raster graphics) with which they are filled. Layout objects have attributes. A block, for instance, beside its identifier and type, has two coordinates, two dimensions and a list of content portion identifiers.

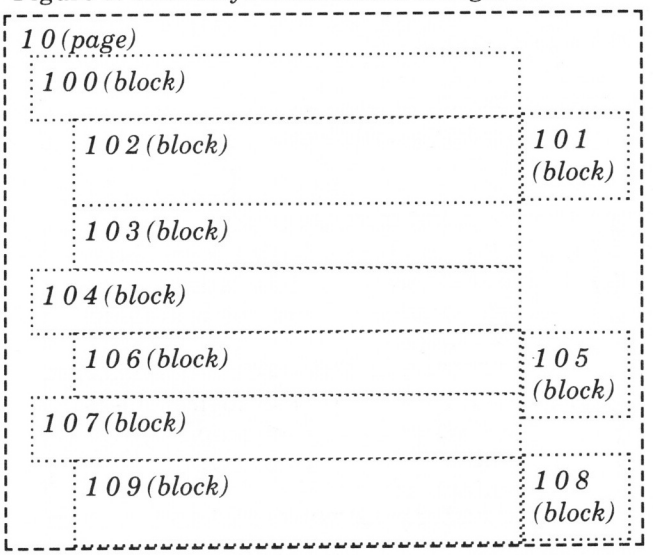
A possible layout structure for our sample document is represented in Figure 5: a set of rectangular blocks owned by a page owned by the document layout root. There are no intermediate frames, but this is purely arbitrary; another ODA application might decide to place a frame around the paragraphs under each title in our example. ODIF assigns the identifier *1* to the layout root, and tracks objects by their owner's identifier and a suffix starting from *0*; hence the numbering scheme shown.

5.2. *Logical Structure*

The logical structure is a hierarchy of objects with a *document logical root* at the top, intermediate levels of *composite logical objects*, and one bottom level of *basic logical objects*. Composite logical objects are optional; the simplest logical stucture consists of the logical root and a basic logical object. Like layout blocks, basic logical objects point to content portions of a single type.

ODA, like SGML, does not attach a meaning to the logical objects. It does not say, for instance, that the top composite logical object represents a chapter and the bottom basic logical object a paragraph or a figure. Particular ODA applications might choose to attach a meaning to logical objects, perhaps for

Figure 5: ODA Layout Structure of Figure 1



the benefit of the user interface, but that meaning cannot be understood by other ODA applications because it is not standardized.

A logical structure for the sample document might be the following. At the top is the document logical root, to which ODIF assigns the identifier *3*. The next level level are composite logical objects, identified as *3 0*, *3 1* and *3 2*. We could call each a "Section", to take advantage of the optional *user visible name* attribute, which may be used in the user interface of an ODA application. The last level is made up of basic logical objects; let's give them the name "Paragraph" when they contain text and "Illustration" otherwise. Figure 6 represents the logical structure and attributes just described.

5.3. *Content*

Document content means characters, raster graphics, geometric graphics, and other data types to be introduced at a later date. These data types are or will be described in *Content Architectures*, separate sections of the standard which generally refer back to other standards. The Geometric Graphics Content Architecture, for instance, is based on the Computer Graphics Metafile Standard [ISO86b].

In the ODIF stream, the content is parcelled out in portions with a logical identifier or a layout identifier or both. These identifiers are present when there is a layout block or a basic logical object pointing to the content portion.

Figure 6: Logical Structure of Figure 1

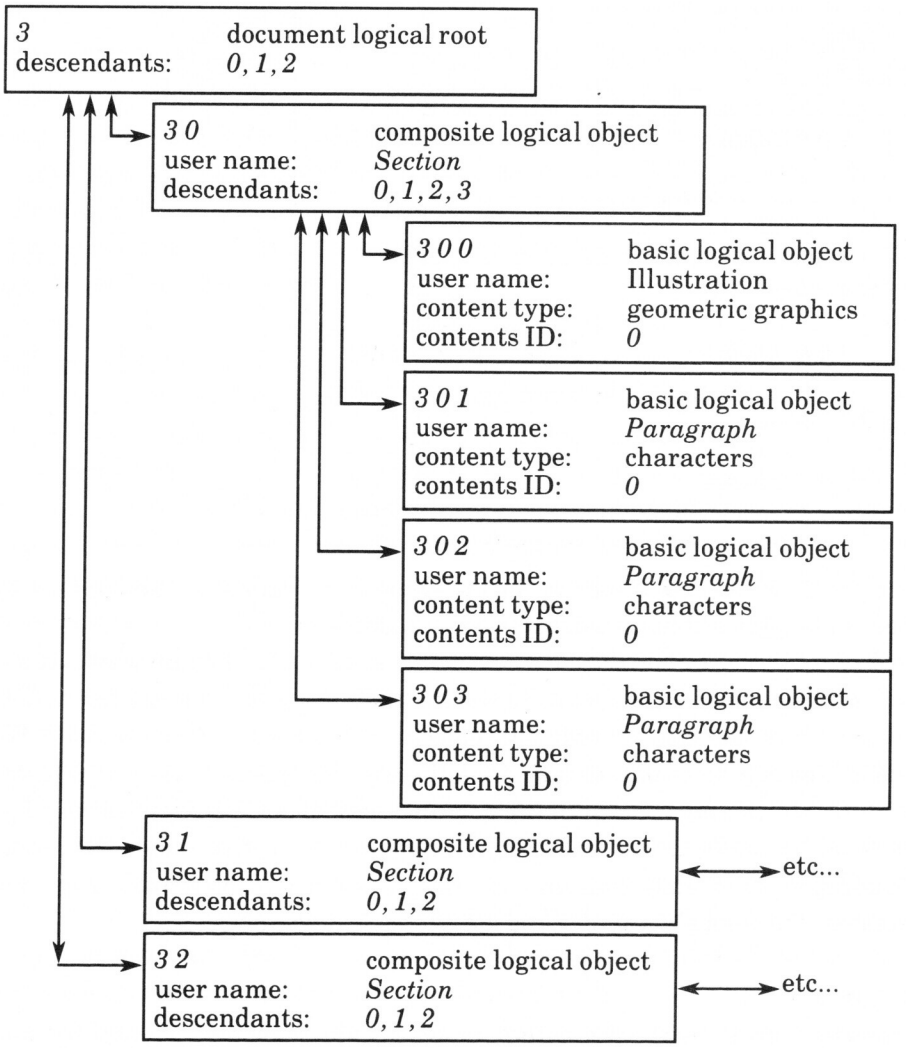


Figure 7 gives a legible representation of the encoding of the content portions of the sample document.

Within each type of content, there are additional structures to govern both the layout and the logical organization of the content's elements. For instance, character text has codes to specify typeface changes, superscripts, hyphenation points and the like. Readers of the standard will note that the part on character content architecture reverts back to the typewriter model of DIF and DCA: it is replete with Partial Line Up, Line Feed, Backspace, Reverse Presentation Direction, etc.

Figure 7: Content Portions of Figure 1 (legible representation)

logical content identifier:	*3 0 0 0*
layout content identifier:	*1 0 1 0*
coding:	geometric graphics
content value:	sequence of VDM instructions for the smiling face picture
logical content identifier:	*3 0 1 0*
layout content identifier:	*1 0 0 0*
coding:	character text
content value:	text *1. The Beginning.* intermingled with control codes for typeface, margins, etc.
logical content identifier:	*3 0 2 0*
layout content identifier:	*1 0 2 0*
coding:	character text
content value:	text *John ... this morning.* & codes
logical content identifier:	*3 0 3 0*
layout content identifier:	*1 0 9 0*
coding:	character text
content value:	text *It was too good to be true....* & codes
etc...	
logical content identifier:	*3 2 2 0*
layout content identifier:	*1 0 9 0*
coding:	character text
content value:	text *... Jane said.... mortgage.* & codes

5.4. *Assessment*

The above section does not discuss many other aspects of ODA, such as generic and specific structures, document and application profiles, styles, objects and document classes. ODA appears to be a heavy standard (lengthy and difficult to understand), but is it heavy-duty? This will not be known until there are enough implementations to test it in the real world.

6. Conclusion

In the matter of document interchange format, much work remains to be done on two fronts. First, we need a complete interchange standard. Existing products like DIF, DCA and SGML have already rendered numerous services, within the constraints for which they were designed. But some of the existing

products cannot interchange composite documents, others require a team of programmers at the receiving end, and the jury is still out in the case of ODA.

Second, we need a clear understanding of what application programs should do with objects and structures they do not support. Discarding the information is not the solution. There can be no complete interchange until we insure round-trip exchange without loss.

References

[1] American National Standards Institute (1979). *Additional Controls for Use with American National Standard Code for Information Interchange.* ANSI X3.64.

[2] American National Standards Institute (1983). *Text Information Interchange in Page Image Format (PIF).* ANSI X3.98.

[3] IBM (1983). *Document Content Architecture: Revisable-Form-Text Reference.* SC23-0758.

[4] IBM (1985). *Document Content Architecture: Final-Form-Text Reference.* SC23-0757.

[5] International Organization for Standardization (1983). *Additional Control Functions for Character Imaging Devices.* ISO IS 6429.

[6] International Organization for Standardization (1986). *Standard Generalized Markup Language.* ISO DIS 8879.

[7] International Organization for Standardization (1986). *Computer Graphics Metafile.* ISO IS 8632.

[8] International Organization for Standardization (1987). *Office Document Architecture.* ISO DIS 8613.

[9] Joloboff, V. (1986). *Trends and Standards in Document Representation.* Proceedings of the Conference on Text Processing and Document Manipulation, Nottingham, U.K..

[10]Knoerdel, J. & Ward Watkins, S. (1984). *Document Interchange Format.* National Bureau of Standards, NBSIR 84-2836.